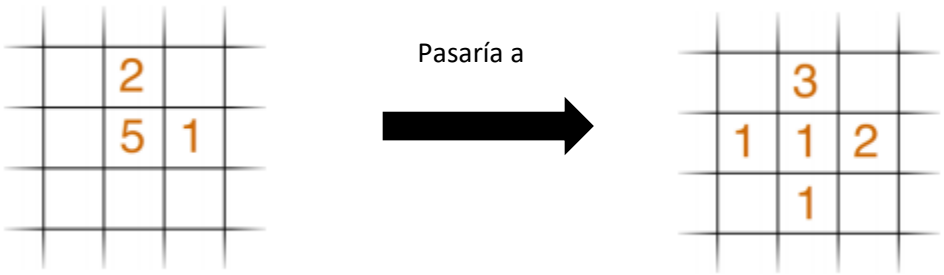
Informe Tarea 1 CC3001

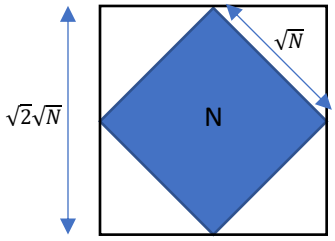
Franco Zautzik

Se presento un modelo para imitar el comportamiento de pilas de granos de arena, en este el área en la que se esparcían los granos de arena era representada como un tablero. Adicionalmente, cada vez que se apilaba 4 o mas granos en una de las posiciones del tablero los granos se esparcían equitativamente entre los puntos cardinales de esta. Es decir:



A partir de lo anterior se debía crear un programa en java llamado *PilaArena* en el cual un usuario ingresara un numero de granos de arena y luego estos fueran puestos en el centro del tablero para luego seguir la regla anterior. El programa debía entregar una ventana en la cual se mostrase el resultado final y que imprimiese la cantidad de aplicaciones de la regla.

Para la creación del tablero primero se obtuvo el tamaño del tablero, para esto se calculó una cota superior. La intuición para obtener lo anterior fue que el esparcimiento de los granos sería menor a un rombo cuadrado de área N por lo tanto una cota superior para el esparcimiento era necesariamente $\sqrt{2}\sqrt{N}$.

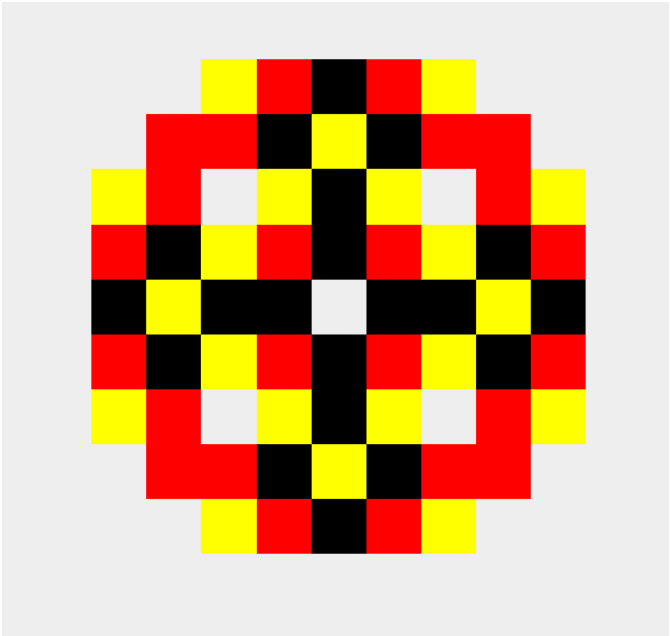


Probando con distintas constantes junto al termino $\sqrt{N}$ se descubrió que la constante 1 bastaba y así el tablero resulto del tamaño $\sqrt{N} \times \sqrt{N}$.

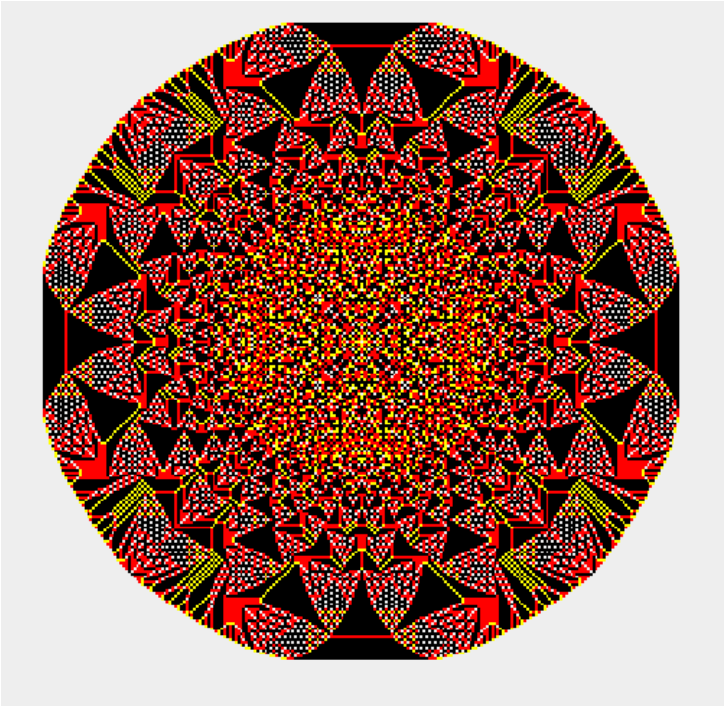
Luego de establecer lo anterior se creo un programa en el cual se creaba una matriz, que representaría el tablero, y luego un ciclo iterativo donde se recorría toda la matriz varias veces para aplicar la regla donde correspondiera hasta que se alcanzara la estabilidad. De esto resulto el siguiente ciclo:

```
while (t == true ){
  t = false;
  for (int i = 0; i < size; i++){
     for(int j = 0; j < size; j++){
       if (tablero[i][j] >= 4){
          t = true;
          veces += 1;
          tablero[i][j] -= 4;
          tablero[i+1][j] += 1;
          tablero[i-1][j] += 1;
          tablero[i][j+1] += 1;
          tablero[i][j-1] += 1;
       }
```

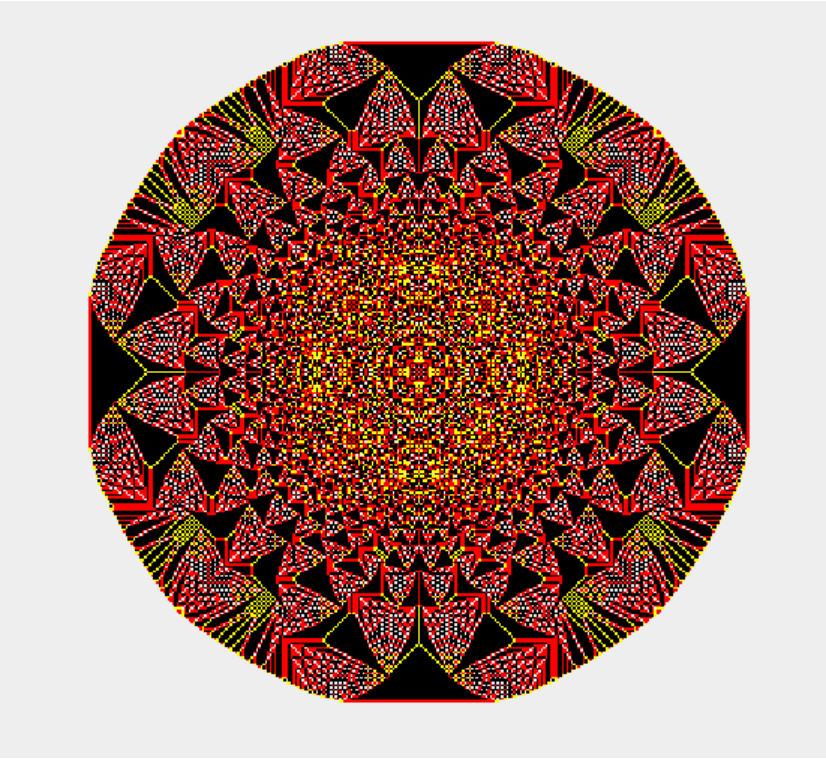
Como respuesta del programa (arrojando la matriz en forma de ventana) se obtuvo lo siguiente:



Para N = 128 con 348 aplicaciones de la regla.



Para N = 100000 con 178641503 aplicaciones de la regla.



Para N = 150000 con 398995983 aplicaciones de la regla.

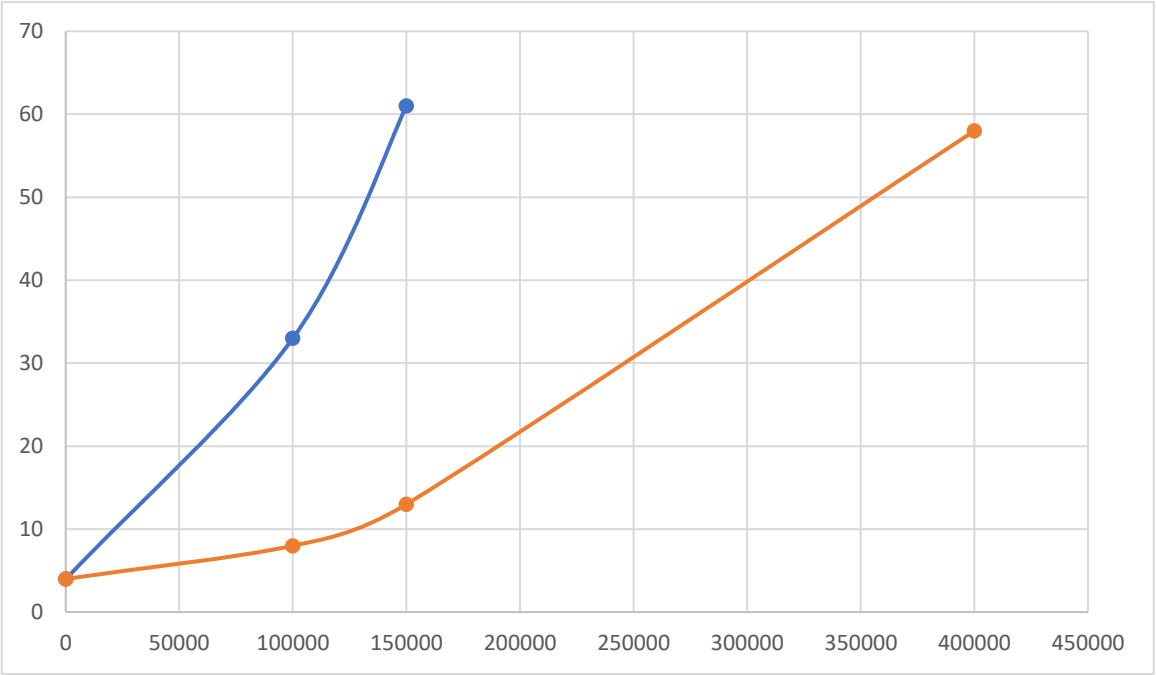
En una segunda etapa se busco mejorar la eficiencia del programa, para ello se cambió el método para esparcir los granos de arena. En el programa anterior se esparcía de a cuatro los granos de arena en caso de que hubiera exceso de granos de arena, sin embargo ¿Qué ocurre si se tiene múltiplos de cuatro granos de arena? Claramente seria mas eficiente esparcir de manera equitativa la cantidad de granos que haya. Para hacer lo anterior solo se tenia que editar levemente el código en la parte de la iteración sobre el tablero:

```
while (t == true ){
    t = false;
    for (int i = 0; i < size; i++){
        for(int j = 0; j < size; j++){
            if (tablero[i][j] >= 4){
                t = true;
                derrumbes = tablero[i][j]/4;
                veces += 1;
                tablero[i][j] -= 4*derrumbes;
                tablero[i+1][j] += derrumbes;
                tablero[i-1][j] += derrumbes;
                tablero[i][j+1] += derrumbes;
                tablero[i][j-1] += derrumbes;
            }
```

Con esta pequeña modificación del programa se reduciría bastante la cantidad de iteraciones para esparcir los granos pues se reducía la cantidad de veces que era necesario aplicar la regla.

Los resultados entregados fueron los mismos, sin embargo, el tiempo de demora del programa difirió importantemente para los valores mayores a 10000.

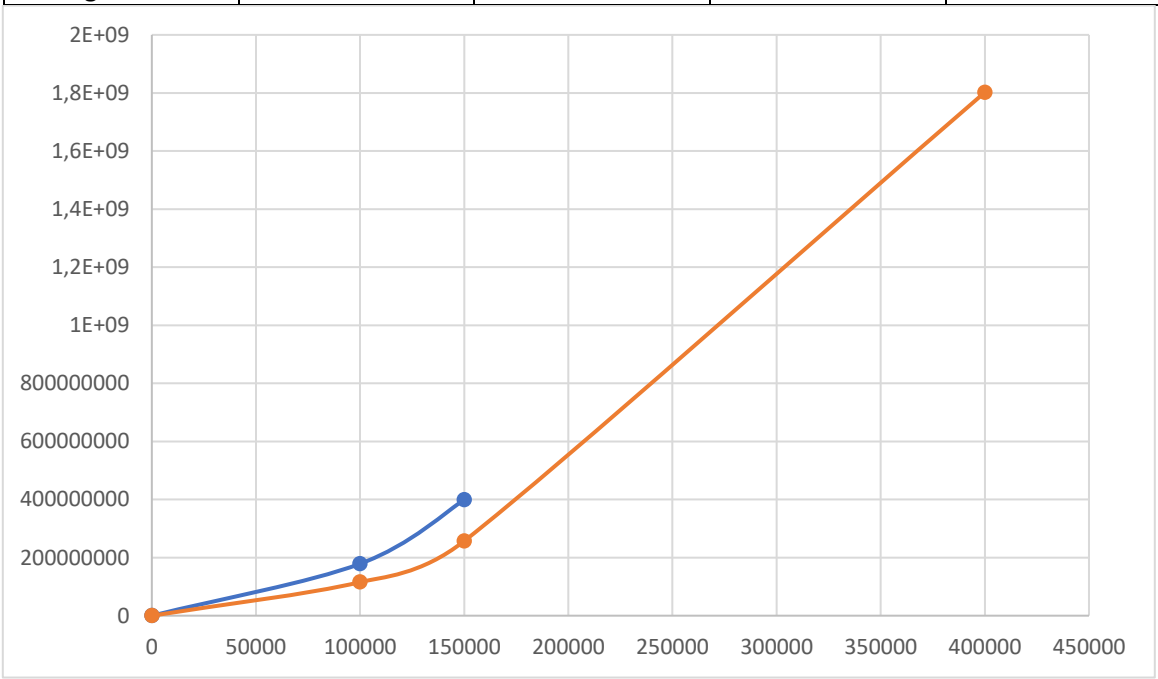
	N = 128	N = 100000	N = 150000	N = 400000
Tiempo de demora Parte 1 [s]	4	33	61	No hubo respuesta
Tiempo de demora Parte 2 [s]	4	8	13	58



En la tabla se puede notar que los tiempos de respuesta del programa aumentan mucho más en el caso del código no optimizado. Viendo un gráfico del tiempo y la cantidad de granos es posible notar que ambos aumentan de manera no lineal y que el código optimizado tiene una pendiente mucho menos pronunciada.

En cuanto a la relación entre el numero de granos y la cantidad de aplicaciones de la regla de distribución se tienen los valores tabulados a continuación:

	N = 128	N = 100000	N = 150000	N = 400000
Numero de aplicaciones de la regla Parte 1	348	178641503	398995983	No hubo respuesta
Numero de aplicaciones de la regla Parte 2	227	115491346	256991021	1802088651



Nuevamente se observa la misma conducta anterior. El código optimizado realiza muchas menos aplicaciones de la regla que el programa original.

La similitud en el aspecto de ambos gráficos no es coincidencia pues claramente están fuertemente relacionados. El numero de aplicaciones de la regla nos muestra la cantidad de veces que el programa tuvo que ir a través del *if* en la línea 28 del código.

```
23          while (t == true ){
24              t = false;
25              for (int i = 0; i < size; i++){
26                  for(int j = 0; j < size; j++){
27                      if (tablero[i][j] >= 4){
28                          t = true;
```

Si la cantidad de veces que el programa tuvo que atravesar el *if* disminuyó eso implica que se tuvieron que realizar menos acciones y en consecuencia se ocupó menos tiempo.

Para concluir, otra manera en la cual se podría optimizar el código es guardando los índices en los cuales cayeron los derrumbes de granos para luego iterar solo sobre aquellos índices. Esto sería mucho más optimo dado que en el código actual el programa itera sobre todo el tablero, aunque se sabe que es inútil que revise casillas que todavía no tienen granos.